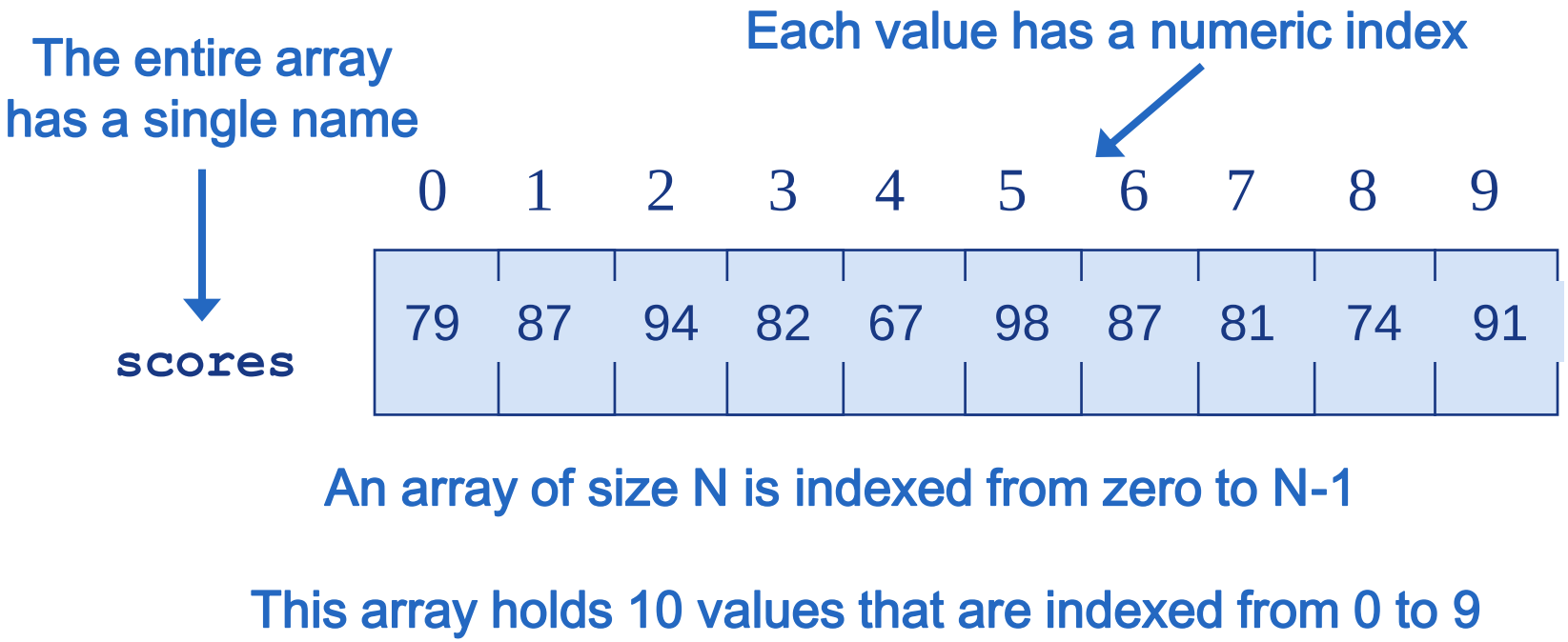
Java - Arrays

Presented by Sagar
Java Consultant

Arrays

- An *array* is an ordered list of values



An array of size N is indexed from zero to N-1

This array holds 10 values that are indexed from 0 to 9

Arrays . . .

- A particular value in an array is referenced using the array name followed by the index in brackets
- For example, the expression

```
scores[2]
```

  refers to the value **94** (the 3rd value in the array)

- That expression represents a place to store a single integer and can be used wherever an integer variable can be used

Initializer list

- An *initializer list* can be used to instantiate and fill an array in one step
- The values are delimited by braces and separated by commas
- Examples:

```
int[] units = {147, 323, 89, 933, 540,
               269, 97, 114, 298, 476};


char[] letterGrades = {'A', 'B', 'C', 'D', 'F'};
```

Arrays – elements, index

- For example, an array element can be assigned a value, printed, or used in a calculation:

```
scores[2] = 89;

scores[first] = scores[first] + 2;

mean = (scores[0] + scores[1])/2;

System.out.println ("Top = " + scores[5]);
```

Arrays – elements, index

- The values held in an array are called ***array elements***
- An array stores multiple values of the same type – the *element type*
- The element type can be a primitive type or an object reference
- Therefore, we can create an array of integers, an array of characters, an array of **`String`** objects, an array of **`Coin`** objects, etc.
- In Java, the array itself is an object that must be instantiated

Arrays – declaration

- The **`scores`** array could be declared as follows:

```
int[] scores = new int[10];
```

- The type of the variable **`scores`** is **`int[]`** - an array of int type)
- Note that the array type does not specify its size, but each object of that type has a specific size
- The reference variable **`scores`** is set to a new array object that can hold 10 integers
- An array is an object, therefore all the values are initialized to default ones (**zero**)

Arrays – elements, index

- Some other examples of array declarations:

```
float[] prices =
                          new float[500];

boolean[] flags;
     flags = new boolean[20];

  char[] codes = new char[1750];
```

Arrays – elements, index

- The iterator version of the **`for`** loop can be used when processing array elements

```
for (int score : scores)
    System.out.println (score);
```

- **This is only appropriate when processing all array elements from top (lowest index) to bottom (highest index)**

Arrays – Example

```
final int LIMIT = 15, MULTIPLE = 10;

    int[] list = new int[LIMIT];

    //  Initialize the array values
    for (int index = 0; index < LIMIT; index++)
            list[index] = index * MULTIPLE;

    list[5] = 999;  // change one array value

    //  Print the array values
    for (int value : list)
            System.out.print (value + "  ");
```

Arrays – Size, exceeding index limit

- Once an array is created, it has a fixed size
- An index used in an array reference must specify a valid element
- That is, the index value must be in range 0 to N-1
- The Java interpreter throws an **ArrayIndexOutOfBoundsException** if an array index is out of bounds
- This is called automatic *bounds checking*

Bounds Checking

- For example, if the array **codes** can hold 100 values, it can be indexed using only the numbers 0 to 99
- If the value of **count** is 100, then the following reference will cause an exception to be thrown:

```
System.out.println (codes[count]);
```

problem

```
for (int index=0; index <= 100; index++)
    codes[index] = index*50;
```

Arrays – length – instance constant

- Each array object has a public constant(instance constant) called **`length`** that stores the size of the array
- It is referenced using the array name:

```
scores.length
```

- Note that **`length`** holds the number of elements, not the largest index

Char type

```
String st = "abcd";
for(int i =0; i < st.length (); i++ ) {
        char c = st.charAt (i);
        System.out.print(c);
        System.out.print("  ");
        System.out.print((int) c);
        System.out.print("  ");
        System.out.println(c - 'a');
}
```

a 97 0

b 98 1

c 99 2

Float type

- The brackets of the array type can be associated with the element type or with the name of the array
- Therefore the following two declarations are equivalent:

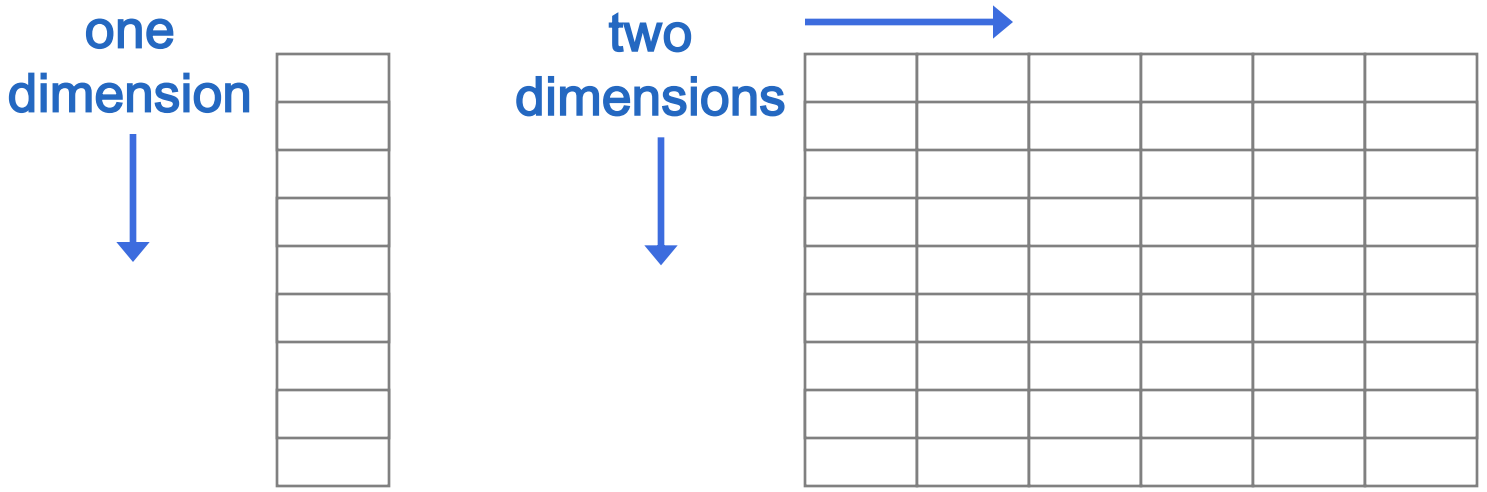
```
float[] prices;

float prices[];
```

- The first format generally is more readable and should be used

Two Dimensional Arrays

- A *one-dimensional array* stores a list of elements
- A *two-dimensional array* can be thought of as a table of elements, with rows and columns



Two Dimensional Arrays …

- To be precise, in Java a two-dimensional array is an array of arrays
- A two-dimensional array is declared by specifying the size of each dimension separately:

```
int[][] scores = new int[12][50];
```

- A array element is referenced using two index values:

```
value = scores[3][6]
```

- The array stored in one row can be specified using one index

Two Dimensioned Array - Example

```
public static void main (String[] args) {
    int[ ][ ] table = new int[5][10];

    // Load the table with values
    for (int row=0; row < table.length; row++)
        for (int col=0; col < table[row].length; col++)
            table[row][col] = row * 10 + col;

    // Print the table
    for (int row=0; row < table.length; row++) {
        for (int col=0; col < table[row].length; col++)
            System.out.print (table[row][col] + "\t");
        System.out.println();
    } // end of  for
} // end of main()
```

Result

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	21	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49


return
THANK
YOU
control